appendix *Answers to exercises*

Chapter 2

> *Can you tell if a hash function provides hiding and binding if used as a commitment scheme?*

A hash function is *hiding* thanks to the pre-image resistance property; that is, if your input is random enough so that no one can guess it. To fix that, you can generate a random number and hash it with your input, and later, you can reveal both your input and the random number to *open* your commitment. A hash function is *binding* thanks to the second pre-image resistance property.

> *By the way, there is no way this string represents 256 bits (32 bytes), right? How is this secure then?*

We don't care about collision resistance. We only care about second pre-image resistance. Thus, we can truncate the digest to reduce its size.

> *Can you guess how the Dread Pirate Roberts (the pseudonym of Silk Road's webmaster) managed to obtain a hash that contains the name of the website?*

Dread Pirate Roberts created a lot of keys until one ended up hashing to that cool base32 representation. Facebook did the same and is accessible from facebookcorewwwi.onion (https://facebook.com/notes/protect-the-graph/making-connections-to-facebook-more-secure/1526085754298237). These are called *vanity addresses*.

Chapter 3

Can you figure out how a variable-length counter could possibly allow an attacker to forge an authentication tag?

By observing the following message, where || represents string concatenation, `MAC(k, "1" || "1 is my favorite number")`, an attacker can forge a valid authentication tag for the eleventh message, `MAC(k, "11" | " is my favorite number")`.

Caution: not all MACs are PRFs. Can you see why?

Imagine that the following function is a valid MAC and PRF: `MAC(key, input)`, then is the following function a valid MAC? `NEW_MAC = MAC(key, input) || 0x01`? Is it a valid PRF? It is a valid MAC as it prevents forgery, but it's not a valid PRF, as you can easily distinguish the output from a totally random string (because the last byte is always set to 1).

Chapter 6

Using the same shared secret with everyone would be very bad; can you see why?

If I can encrypt messages to you with this shared secret, I can also decrypt messages from other people.

Do you see why you can't use the key exchange output right away?

Remember what you've learned in chapter 5 on key exchanges. In (FF)DH, calculations happen modulo a large prime number p. Let's take a small prime number as example, 65,537. In hexadecimal, our p is written as `0x010001`, and in binary, it is written as `0000 0001 0000 0000 0000 0001`. In binary, notice the zeros preceding the first one because we represent our number in bytes (multiple of 8 bits).

If you understand modular arithmetic, you know that numbers modulo this prime p will never be larger, meaning that the first 7 bits will always be set to 0. In addition, the eighth bit will most often be set to 0 rather than 1. This is not *uniformly random*. Ideally, every bit should have the same probability of being set to 1 or to 0.

Chapter 7

As you saw in chapter 3, authentication tags produced by MACs must be verified in constant time to avoid timing attacks. Do you think we need to do the same for verifying signatures?

No. This is because the verification of an authentication tag involves a secret key. Verifying a signature only involves a public key and, thus, does not need to be verified in constant time.

Chapter 8

Imagine for a minute that mixing different sources of entropy was done by simply XORing them together. Can you see how this might fail to be contributory?

A backdoored source of entropy could set its output as the XOR of all the other sources of entropy, effectively canceling all entropy to 0.

Signature schemes like BLS (mentioned in figure 8.5 and in chapter 7) produce unique signatures, but this is not true for ECDSA and EdDSA. Do you see why?

In ECDSA, the signer can choose different nonces to produce different signatures for the same key pair and message. While EdDSA is a signature algorithm that deterministically derives the nonce based on the message to be signed, this does not mean that the signer cannot use any nonce if they so choose.

Chapter 9

A compromise of the server's private key at some point in time would be devastating as MITM attackers would then be able to decrypt all previously recorded conversations. Do you understand how this can happen?

The attacker would then be able to rewind history and impersonate the server at the time the handshake was performed. Indeed, the attacker now has the server's private key. All the other information to perform the key exchange and derive the posthandshake symmetric keys is public.

The values `signatureAlgorithm` *and* `signatureValue` *are not contained in the actual certificate,* `tbsCertificate`. *Do you know why?*

The Certificate Authority (CA) needs to sign the certificate, which leads to a paradox: the signature cannot be part of the signature itself. The CA must, thus, append the signature to the certificate. Other standards and protocols might use different techniques. For example, you could include the signature as part of `tbsCertificate` and pretend that it is made of all 0s when you sign or verify the certificate.

Chapter 10

Do you know why the email's content is compressed before it is encrypted and not after?

A ciphertext is indistinguishable from a random string according to the definition of a cipher. Due to this, compression algorithms are incapable of finding patterns to efficiently compress encrypted data. For this reason, compression is always applied before encryption.

Can you think of an unambiguous way of signing a message?

One line: authenticate the context. A way to do this is to include both the sender and the recipient's names and their public keys in the signature and then encrypt that.

Chapter 11

Sometimes applications attempt to fix the issue of the server learning about the user passwords at registration by having the client hash (perhaps with a password hash) the password before sending it to the server. Can you determine if this really works?

Client-side hashing alone does not work as the infamous pass-the-hash attack showed (https://en.wikipedia.org/wiki/Pass_the_hash); if the server stores Alice's hashed password directly, then anyone who steals it can also use it as a password to authenticate as Alice. Some applications perform both client-side hashing and server-side hashing, which, in this case, can perhaps prevent an active attacker from knowing the original password (although an active attacker might be able to disable client-side hashing by updating the code of the client application).

Imagine a protocol where you have to enter the correct 4-digit PIN to securely connect to a device. What are the chances to pick a correct PIN by just guessing?

That's 1 out of 10,000 chances to correctly guess something. You'd be happy if you were playing Lotto with these odds.

index

Numerics

A

D

E

I

J

K

O

P

Q

R

S

T

U

V

W

X

Z

CRYPTOGRAPHY/SECURITY

Real-World Cryptography

David Wong

Cryptography is the essential foundation of IT security. To stay ahead of the bad actors attacking your systems, you need to understand the tools, frameworks, and protocols that protect your networks and applications. This book introduces authentication, encryption, signatures, secret-keeping, and other cryptography concepts in plain language and beautiful illustrations.

Real-World Cryptography teaches practical techniques for day-to-day work as a developer, sysadmin, or security practitioner. There's no complex math or jargon: Modern cryptography methods are explored through clever graphics and real-world use-cases. You'll learn building blocks like hash functions and signatures; cryptographic protocols like HTTPS and secure messaging; and cutting-edge advances like post-quantum cryptography and cryptocurrencies. This book is a joy to read—and it might just save your bacon the next time you're targeted by an adversary after your data.

What's Inside

- Implementing digital signatures and zero-knowledge proofs
- Specialized hardware for attacks and highly adversarial environments
- Identifying and fixing bad practices
- Choosing the right cryptographic tool for any problem

For cryptography beginners with no previous experience in the field.

David Wong is a cryptography engineer. He is an active contributor to internet standards including Transport Layer Security.

Register this print book to get free access to all ebook formats.
Visit https://www.manning.com/freebook

"A staggeringly comprehensive review of the state of modern cryptography. Essential for anyone getting up to speed in information security."
—Thomas Doylend
Green Rocket Security

"Covers all the important parts of applied cryptography. A must read for every Java developer."
—Harald Kuhn, TurnFriendly

"An in-depth introduction to cryptography, covering hot topics like blockchain and quantum computing."
—Gábor László Hajba, ProLion

"A best-in-category book that takes you all the way from curious novice to confident practitioner."
—William Rudenmalm
Creandum

Free eBook
See first page

MANNING $59.99 / Can $79.99 [INCLUDING eBOOK]

ISBN: 978-1-61729-671-0

55999

9 781617 296710